Systems Design Document

# Introduction

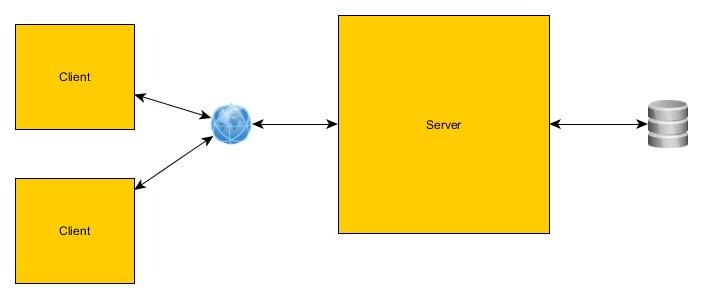
This document is a high-level, language-independent description of the system design. It is not intended to replace a low-level, detailed technical design document.

# Proposed Software Architecture

## Overview

The system will be comprised of two major parts: a server and a client. The server shall be responsible for all persistence and implement the game rules, while the client shall be responsible for the end-user look-and-feel and little else.

## Subsystem Descriptions



### Server

The server shall send and receive information via a strictly-defined JSON API. The server shall maintain the state of each game, as well as the state of the lobby. The server shall receive information about player actions from the clients and respond with the updated state of the game after that action was performed, or an error if the action is invalid. The server shall maintain user information for each player, including the master count of chips in the system. The server shall also be responsible for validation of game state and player actions.

### Client

The client shall present a user interface to the user that is clean and discoverable. The client shall allow the player to make their choices, and send the information to the server using the JSON API. The client shall update when the server pushes out a new game state. The client shall present information about the game state to the player.

## Persistent Data Management

The server shall maintain persistent data about users and their chip count in a SQL Lite database.

The database shall contain a table to manage User information. Each User entity shall contain an ID, a Username, a Password (stored as per the Security section below), an Email, and a current chip count.

The database shall also maintain a list of currently valid sessions. Each session shall reference one User entity, and contain both a token and an expiration date.

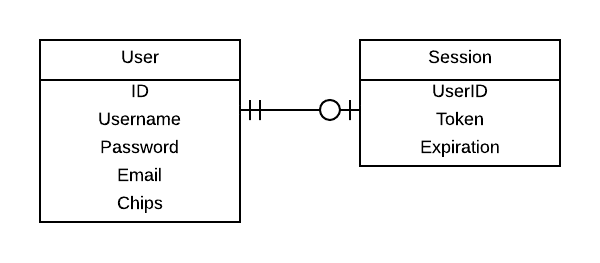


Figure Entity-Relationship Diagram

## Access Control and Security

### Password Selection

The user shall be able to select a username and a password upon registration, as well as an email address. The user shall be able to change their password at will. An email can be triggered to be sent to the user to reset their password if need be.

### Transport Layer

The messaging between client and server shall be built on SSL, to protect the integrity of user passwords, hands, and access tokens.

### Password Storage

The password shall be stored encrypted with the bcrypt key derivation function, and shall not be able to be restored in plaintext. No additional salting is required with a correct implementation of bcrypt.

### Authentication Token

Upon receiving a successful username and password combination, the server shall reply with a one-use token that will expire in 24 hours. The client shall send that token along with every subsequent message; if a message other than a login attempt is sent without a token, the server shall reject it.

## Messaging Protocol

Messages from the client to the server and from the server to the client shall consist of JSON using the UTF-8 character encoding.

The following messages can be sent:

### User Creation Attempt

A login attempt is sent from the client to the server. It consists of the following key-value pairs:

* **Action:** always “Create User”
* **Parameters:** An object with the following key-value pairs:
  + **Username:** A string value containing the username
  + **Email:** A string value containing the email
  + **Password:** A string value containing the desired password

### Authentication Attempt

A login attempt is sent from the client to the server. It consists of the following key-value pairs:

* **Action:** always “Authenticate”
* **Parameters:** An object with the following key-value pairs:
  + **Username:** A string value containing the username
  + **Password:** A string value containing the password

### Player Action

A player action message is sent when the user is trying to perform an action on the client that should be handled by the server. It consists of the following key-value pairs:

* **Token:** The token last provided by the server
* **Action:** Can be “Join”, “Bet”, “Fold”, or “Quit”
* **Parameters:** An object, specified below.

#### Join Action Parameters

The Parameters object for an action “Join” consists of the following key-value pairs:

* **GameID:** An ID for a game to be joined.

#### Bet Action Parameters

* **Amount:** A numeric amount of chips to bet
* **All**-**in:** Whether the player has gone all-in. Valid values are “true” or “false”

#### Fold Action Parameters

* **Quit:** Whether the player is also quitting the game table. Valid values are “true” or “false”

#### Quit Action Parameters

The parameters object for a Quit action is always empty.

### Authentication Success

An authentication success is sent from the server to the client upon receipt of an Authentication Attempt that results in a password match or a User Creation Attempt that did not result in an error. It consists of the following key-value pairs:

* **Message:** “Authentication Success”
* **Token:** A generated token used for identification in future messages.

### Error

An error message is sent from the server to the client when an error has occurred. It consists of the following key-value pairs:

* **ErrorCode:** A numeric code explaining the error. The possible errors shall be documented at a later date.
* **ErrorMessage:** A string describing the error. This should be safe to display to the user.

### Game State

A game state message is sent from the server to the client. It is an unsolicited message, and the only unsolicited message. It consists of the following key-value pairs:

* **Type:** Valid values are “Lobby” or “Game”
* **State:** An object, specified below.

#### Lobby State Object

A Lobby State object consists of the following key-value pairs:

* **Games:** An array of objects, each consisting of the following key-value pairs:
  + **ID:** The ID of the game.
  + **Open:** Whether this game can be joined. Valid values are “true” and “false”
  + **Players:** An array of objects, each consisting of the following key-value pairs:
    - **Username:** The name of the player
    - **Avatar:** A string containing a URL to the user’s Gravitar image
    - **Chips:** A numeric value representing the user’s current chip count

#### Game State Object

A Game State object consists of the following key-value pairs:

* **Pot:** A number consisting of the number of chips in the table’s betting pool
* **Dealer:** A username of the player that is currently the Dealer this round
* **Actor:** A username of the player that can currently take actions.
* **TableCards:** An array of 0, 3, 4, or 5 card objects
* **You:** An object consisting of the following key-value pairs:
  + **Position:** A numeric value indicating the seat number you have been assigned
  + **Hand:** An array of 0 or 2 card objects
  + **Chips:** A numeric value representing your current chip count
* **OtherPlayers:** An array of objects, each consisting of the following key-value pairs:
  + **Position:** A numeric value indicating the seat number of the player
  + **Username:** The name of the player. Note that “You” is not included in this list.
  + **Avatar:** A string containing a URL to the user’s Gravitar image
  + **Chips:** A numeric value representing the user’s current chip count
* **LastAction:** This is a Player Action message, minus the authentication token, representing the last move made which created this state so that the interface can display it.

#### Card Object

A Card object consists of the following key-value pairs:

* **Value:** A string representing the value of the card. Valid values are the numbers 2-10, as well as “A”, “J”, “K”, “Q”.
* **Suit:** A string representing the suit of the card. Valid values are “Hearts”, “Diamonds”, “Spades”, “Clubs”